FOUNDATION-0.2b: Agent State Tables - PART 1 (Code)

🎯 CONTEXT

Phase: FOUNDATION (Week 1 - Day 1 Afternoon)
Component: Agent State & Configuration Tables
Estimated Time: 20 min AI execution + 10 min verification
Complexity: MEDIUM
Risk Level: LOW
Files: Part 1 of 2 (Code implementation)

📦 DEPENDENCIES

Must Complete First:

- **FOUNDATION-0.2a:** Core Database Schema ✅ COMPLETED & VALIDATED
  - All 6 core tables created
  - 13/13 tests passing
  - Seed data loaded

Required Services Running:

bash

```bash
# Verify all services are healthy
cd ~/optiinfra
make verify

# Expected output:
# PostgreSQL... ✅ HEALTHY
# ClickHouse... ✅ HEALTHY
# Qdrant... ✅ HEALTHY
# Redis... ✅ HEALTHY
```

Database State:

bash

```
# Verify core tables exist
psql postgresql://optiinfra:password@localhost:5432/optiinfra -c "\dt"

# Expected: customers, agents, events, recommendations, approvals, optimizations
```

🎯 OBJECTIVE

Extend the database schema with **agent-specific state and configuration tables**. These tables enable:

1. **Agent Configuration** - Store agent settings, parameters, thresholds
2. **Agent State** - Track real-time agent state (active workflows, locks, status)
3. **Agent Capabilities** - Detailed capability definitions and versions
4. **Agent Metrics** - Agent-level performance metrics

What This Enables:

- ✅ Store agent configuration (KV cache settings, thresholds, etc.)
- ✅ Track agent state (workflows in progress, locks)
- ✅ Version capabilities (each agent can have multiple capability versions)
- ✅ Monitor agent health metrics
- ✅ Enable agent coordination (orchestrator needs this!)

Success Criteria:

✅ 4 new tables created (agent_configs, agent_states, agent_capabilities, agent_metrics)
✅ Alembic migration working (002_agent_state_tables.py)
✅ All foreign keys to agents table working
✅ Seed data for test agents
✅ 10+ tests passing
✅ Documentation complete

Failure Signs:

❌ Migration fails
❌ Foreign key constraints broken
❌ Cannot insert test data

🗺️ TECHNICAL SPECIFICATION

Database Schema Design

📋
✓

```
┌────────────────────────────┐
│    agents       │  (from 0.2a - existing)
│ (5 test records) │
└─────────────┬──────────────┘
    │
    ├──────────────────────────────────────────────────────┐
    │                     │
    ▼                     ▼
┌──────────────────────────────┐        ┌─────────────────────────────┐
│ agent_configs    │          │ agent_states     │
├──────────────────────────────┤        ├─────────────────────────────┤
│ id (PK)          │          │ id (PK)          │
│ agent_id (FK)    │          │ agent_id (FK)    │
│ config_key       │          │ current_status   │
│ config_value     │          │ active_workflows │
│ config_type      │          │ locks            │
│ description      │          │ last_activity    │
│ created_at       │          │ resource_usage   │
│ updated_at       │          │ created_at       │
└──────────────────────────────┘        │ updated_at       │
                  └──────────────────────────────┘
    │
    ├──────────────────────────────────────────────────────┐
    │                     │
    ▼                     ▼
┌──────────────────────────────┐        ┌─────────────────────────────┐
│ agent_capabilities │        │ agent_metrics    │
├──────────────────────────────┤        ├─────────────────────────────┤
│ id (PK)            │        │ id (PK)          │
│ agent_id (FK)      │        │ agent_id (FK)    │
│ capability_name    │        │ metric_name      │
│ capability_version │        │ metric_value     │
│ enabled            │        │ metric_type      │
│ config             │        │ recorded_at      │
│ created_at         │        │ tags             │
│ updated_at         │        └──────────────────────────────┘
└──────────────────────────────┘
```

File Structure to Create (Part 1)

```
shared/
├── database/
│   ├── models/
│   │   ├── __init__.py        # MODIFY: Export new models ⭐
│   │   ├── core.py            # KEEP: From 0.2a
│   │   └── agent_state.py     # CREATE: Agent state models ⭐
│   └── migrations/
│       └── versions/
│           ├── 001_core_schema.py        # KEEP: From 0.2a
│           └── 002_agent_state_tables.py # CREATE: New migration ⭐
```

💻 IMPLEMENTATION - PART 1

FILE 1: CREATE shared/database/models/agent_state.py

Complete SQLAlchemy models for agent state management:

python

```python
"""
Agent state and configuration models.
Extends the core schema with agent-specific data.
"""
import uuid
from datetime import datetime
from enum import Enum as PyEnum

from sqlalchemy import (
    Column, String, Integer, Float, Boolean, DateTime, Text,
    ForeignKey, Enum, Index, UniqueConstraint
)
from sqlalchemy.dialects.postgresql import UUID, JSONB
from sqlalchemy.orm import relationship
from sqlalchemy.sql import func

from shared.database.models.core import Base


# ============================================================================
# ENUMS
# ============================================================================

class ConfigType(str, PyEnum):
    """Configuration value types"""
    STRING = "string"
    INTEGER = "integer"
    FLOAT = "float"
    BOOLEAN = "boolean"
    JSON = "json"


class AgentStatusDetail(str, PyEnum):
    """Detailed agent status"""
    IDLE = "idle"
    BUSY = "busy"
    PROCESSING = "processing"
    WAITING = "waiting"
    ERROR = "error"
```

```python
class MetricType(str, PyEnum):
    """Metric value types"""
    COUNTER = "counter"
    GAUGE = "gauge"
    HISTOGRAM = "histogram"


# ==========================================================================
# MODELS
# ==========================================================================

class AgentConfig(Base):
    """
    Agent configuration storage.

    Stores key-value configuration for each agent.
    Examples: thresholds, timeouts, feature flags, optimization parameters.
    """
    __tablename__ = "agent_configs"

    id = Column(
        UUID(as_uuid=True),
        primary_key=True,
        default=uuid.uuid4,
        comment="Unique config entry identifier"
    )
    agent_id = Column(
        UUID(as_uuid=True),
        ForeignKey("agents.id", ondelete="CASCADE"),
        nullable=False,
        index=True,
        comment="Agent this config belongs to"
    )
    config_key = Column(
        String(255),
        nullable=False,
        index=True,
        comment="Configuration key (e.g., 'kv_cache_size')"
    )
    config_value = Column(
        Text,
        nullable=False,
        comment="Configuration value (stored as string)"
    )
    config_type = Column(
        Enum(ConfigType, create_type=False, values_callable=lambda x: [e.value for e in x]),
        nullable=False,
        default=ConfigType.STRING,
        comment="Type of configuration value"
    )
    description = Column(
        Text,
        nullable=True,
        comment="Human-readable description of this config"
    )
    created_at = Column(
        DateTime(timezone=True),
        nullable=False,
        server_default=func.now(),
        comment="Config creation timestamp"
    )
    updated_at = Column(
        DateTime(timezone=True),
        nullable=False,
        server_default=func.now(),
        onupdate=func.now(),
        comment="Last update timestamp"
    )

    # Relationships
    agent = relationship("Agent", back_populates="configs")

    # Indexes and Constraints
    __table_args__ = (
        Index("idx_agent_config_agent", "agent_id"),
        Index("idx_agent_config_key", "config_key"),
        Index("idx_agent_config_agent_key", "agent_id", "config_key"),
        UniqueConstraint("agent_id", "config_key", name="uq_agent_config"),
    )

    def __repr__(self):
        return f"<AgentConfig(agent_id={self.agent_id}, key='{self.config_key}', type={self.config_type.value})>"
```

```python
class AgentState(Base):
    """
    Real-time agent state.

    Tracks the current operational state of each agent:
    - Active workflows
    - Resource locks
    - Current status
    - Resource usage
    """
    __tablename__ = "agent_states"

    id = Column(
        UUID(as_uuid=True),
        primary_key=True,
        default=uuid.uuid4,
        comment="Unique state identifier"
    )
    agent_id = Column(
        UUID(as_uuid=True),
        ForeignKey("agents.id", ondelete="CASCADE"),
        nullable=False,
        unique=True,
        index=True,
        comment="Agent this state belongs to (one-to-one)"
    )
    current_status = Column(
        Enum(AgentStatusDetail, create_type=False, values_callable=lambda x: [e.value for e in x]),
        nullable=False,
        default=AgentStatusDetail.IDLE,
        index=True,
        comment="Current detailed status"
    )
    active_workflows = Column(
        JSONB,
        nullable=False,
        default=[],
        comment="List of currently active workflow IDs"
    )
```

```python
locks = Column(
    JSONB,
    nullable=False,
    default={},
    comment="Resource locks held by this agent"
)
last_activity = Column(
    DateTime(timezone=True),
    nullable=False,
    server_default=func.now(),
    index=True,
    comment="Last activity timestamp"
)
resource_usage = Column(
    JSONB,
    nullable=True,
    default={},
    comment="Current resource usage (CPU, memory, etc.)"
)
metadata = Column(
    JSONB,
    nullable=True,
    default={},
    comment="Additional state metadata"
)
created_at = Column(
    DateTime(timezone=True),
    nullable=False,
    server_default=func.now(),
    comment="State creation timestamp"
)
updated_at = Column(
    DateTime(timezone=True),
    nullable=False,
    server_default=func.now(),
    onupdate=func.now(),
    comment="Last update timestamp"
)

# Relationships
agent = relationship("Agent", back_populates="state", uselist=False)
```

```python
    # Indexes
    __table_args__ = (
        Index("idx_agent_state_agent", "agent_id"),
        Index("idx_agent_state_status", "current_status"),
        Index("idx_agent_state_activity", "last_activity"),
    )

    def __repr__(self):
        return f"<AgentState(agent_id={self.agent_id}, status={self.current_status.value})>"


class AgentCapability(Base):
    """
    Agent capability definitions.

    Tracks what each agent can do, with version information.
    Example: Cost Agent can do "spot_migration" v1.2.0
    """
    __tablename__ = "agent_capabilities"

    id = Column(
        UUID(as_uuid=True),
        primary_key=True,
        default=uuid.uuid4,
        comment="Unique capability identifier"
    )
    agent_id = Column(
        UUID(as_uuid=True),
        ForeignKey("agents.id", ondelete="CASCADE"),
        nullable=False,
        index=True,
        comment="Agent this capability belongs to"
    )
    capability_name = Column(
        String(255),
        nullable=False,
        index=True,
        comment="Capability name (e.g., 'spot_migration')"
    )
    capability_version = Column(
    String(50),
    nullable=False,
    default="1.0.0",
    comment="Capability version (semver)"
)
enabled = Column(
    Boolean,
    nullable=False,
    default=True,
    index=True,
    comment="Whether this capability is currently enabled"
)
config = Column(
    JSONB,
    nullable=True,
    default={},
    comment="Capability-specific configuration"
)
description = Column(
    Text,
    nullable=True,
    comment="Human-readable description"
)
created_at = Column(
    DateTime(timezone=True),
    nullable=False,
    server_default=func.now(),
    comment="Capability registration timestamp"
)
updated_at = Column(
    DateTime(timezone=True),
    nullable=False,
    server_default=func.now(),
    onupdate=func.now(),
    comment="Last update timestamp"
)

# Relationships
agent = relationship("Agent", back_populates="capabilities")

# Indexes and Constraints
```

```
    __table_args__ = (
        Index("idx_agent_cap_agent", "agent_id"),
        Index("idx_agent_cap_name", "capability_name"),
        Index("idx_agent_cap_enabled", "enabled"),
        Index("idx_agent_cap_agent_name", "agent_id", "capability_name"),
        UniqueConstraint("agent_id", "capability_name", name="uq_agent_capability"),
    )

    def __repr__(self):
        return f"<AgentCapability(agent_id={self.agent_id}, name='{self.capability_name}', version='{self.capability_versio


class AgentMetric(Base):
    """
    Agent-level performance metrics.

    Time-series metrics for agent performance:
    - Request counts
    - Success rates
    - Processing times
    - Resource usage
    """
    __tablename__ = "agent_metrics"

    id = Column(
        UUID(as_uuid=True),
        primary_key=True,
        default=uuid.uuid4,
        comment="Unique metric identifier"
    )
    agent_id = Column(
        UUID(as_uuid=True),
        ForeignKey("agents.id", ondelete="CASCADE"),
        nullable=False,
        index=True,
        comment="Agent this metric belongs to"
    )
    metric_name = Column(
        String(255),
        nullable=False,
        index=True,
    comment="Metric name (e.g., 'requests_total')"
)
metric_value = Column(
    Float,
    nullable=False,
    comment="Metric value"
)
metric_type = Column(
    Enum(MetricType, create_type=False, values_callable=lambda x: [e.value for e in x]),
    nullable=False,
    default=MetricType.GAUGE,
    comment="Type of metric"
)
tags = Column(
    JSONB,
    nullable=True,
    default={},
    comment="Metric tags/labels"
)
recorded_at = Column(
    DateTime(timezone=True),
    nullable=False,
    server_default=func.now(),
    index=True,
    comment="Metric recording timestamp"
)

# Relationships
agent = relationship("Agent", back_populates="metrics")

# Indexes
__table_args__ = (
    Index("idx_agent_metric_agent", "agent_id"),
    Index("idx_agent_metric_name", "metric_name"),
    Index("idx_agent_metric_recorded", "recorded_at"),
    Index("idx_agent_metric_agent_name", "agent_id", "metric_name"),
    Index("idx_agent_metric_agent_recorded", "agent_id", "recorded_at"),
)
```

```python
    def __repr__(self):
        return f"<AgentMetric(agent_id={self.agent_id}, name='{self.metric_name}', value={self.metric_value})>"
```

✅ **FILE 1 COMPLETE (~400 lines)**

FILE 2: MODIFY shared/database/models/core.py

Add relationships to the Agent model:

Find the `Agent` class and add these new relationships lines:

python

```python
class Agent(Base):
    # ... existing code ...

    # Relationships (ADD THESE NEW LINES)
    events = relationship("Event", back_populates="agent", cascade="all, delete-orphan")
    recommendations = relationship("Recommendation", back_populates="agent", cascade="all, delete-orphan")
    optimizations = relationship("Optimization", back_populates="agent", cascade="all, delete-orphan")

    # NEW RELATIONSHIPS - ADD THESE:
    configs = relationship("AgentConfig", back_populates="agent", cascade="all, delete-orphan")
    state = relationship("AgentState", back_populates="agent", uselist=False, cascade="all, delete-orphan")
    capabilities = relationship("AgentCapability", back_populates="agent", cascade="all, delete-orphan")
    metrics = relationship("AgentMetric", back_populates="agent", cascade="all, delete-orphan")
```

✅ **FILE 2 MODIFIED**

FILE 3: MODIFY shared/database/models/init.py

Export the new models:

python

```python
"""
Database models package.
"""
from shared.database.models.core import (
    Base,
    # Enums
    CustomerPlan,
    CustomerStatus,
    AgentType,
    AgentStatus,
    EventSeverity,
    RecommendationPriority,
    RecommendationStatus,
    ApprovalStatus,
    OptimizationStatus,
    # Models
    Customer,
    Agent,
    Event,
    Recommendation,
    Approval,
    Optimization,
)

# NEW IMPORTS - ADD THESE:
from shared.database.models.agent_state import (
    # Enums
    ConfigType,
    AgentStatusDetail,
    MetricType,
    # Models
    AgentConfig,
    AgentState,
    AgentCapability,
    AgentMetric,
)

__all__ = [
    "Base",
    # Core Enums
    "CustomerPlan",
    "CustomerStatus",
    "AgentType",
    "AgentStatus",
    "EventSeverity",
    "RecommendationPriority",
    "RecommendationStatus",
    "ApprovalStatus",
    "OptimizationStatus",
    # Agent State Enums - NEW
    "ConfigType",
    "AgentStatusDetail",
    "MetricType",
    # Core Models
    "Customer",
    "Agent",
    "Event",
    "Recommendation",
    "Approval",
    "Optimization",
    # Agent State Models - NEW
    "AgentConfig",
    "AgentState",
    "AgentCapability",
    "AgentMetric",
]
```

✅ **FILE 3 MODIFIED**

FILE 4: CREATE shared/database/migrations/versions/002_agent_state_tables.py

Alembic migration to create agent state tables:

python

```
"""
Create agent state tables

Revision ID: 002_agent_state_tables
Revises: 001_core_schema
Create Date: 2025-10-18 18:00:00.000000
"""
from alembic import op
import sqlalchemy as sa
from sqlalchemy.dialects import postgresql

# revision identifiers, used by Alembic.
revision = '002_agent_state_tables'
down_revision = '001_core_schema'
branch_labels = None
depends_on = None


def upgrade():
    """Create agent state tables"""

    # Create ENUM types
    op.execute("""
        CREATE TYPE configtype AS ENUM ('string', 'integer', 'float', 'boolean', 'json');
        CREATE TYPE agentstatusdetail AS ENUM ('idle', 'busy', 'processing', 'waiting', 'error');
        CREATE TYPE metrictype AS ENUM ('counter', 'gauge', 'histogram');
    """)

    # Create agent_configs table
    op.create_table(
        'agent_configs',
        sa.Column('id', postgresql.UUID(as_uuid=True), primary_key=True),
        sa.Column('agent_id', postgresql.UUID(as_uuid=True), sa.ForeignKey('agents.id', ondelete='CASCADE'), nullable=F
        sa.Column('config_key', sa.String(255), nullable=False),
        sa.Column('config_value', sa.Text, nullable=False),
        sa.Column('config_type', postgresql.ENUM(name='configtype'), nullable=False, server_default='string'),
        sa.Column('description', sa.Text, nullable=True),
        sa.Column('created_at', sa.DateTime(timezone=True), nullable=False, server_default=sa.text('now()')),
        sa.Column('updated_at', sa.DateTime(timezone=True), nullable=False, server_default=sa.text('now()')),
    )
```

```
# Create indexes for agent_configs
op.create_index('idx_agent_config_agent', 'agent_configs', ['agent_id'])
op.create_index('idx_agent_config_key', 'agent_configs', ['config_key'])
op.create_index('idx_agent_config_agent_key', 'agent_configs', ['agent_id', 'config_key'])
op.create_unique_constraint('uq_agent_config', 'agent_configs', ['agent_id', 'config_key'])

# Create agent_states table
op.create_table(
    'agent_states',
    sa.Column('id', postgresql.UUID(as_uuid=True), primary_key=True),
    sa.Column('agent_id', postgresql.UUID(as_uuid=True), sa.ForeignKey('agents.id', ondelete='CASCADE'), nullable=F
    sa.Column('current_status', postgresql.ENUM(name='agentstatusdetail'), nullable=False, server_default='idle'),
    sa.Column('active_workflows', postgresql.JSONB, nullable=False, server_default='[]'),
    sa.Column('locks', postgresql.JSONB, nullable=False, server_default='{}'),
    sa.Column('last_activity', sa.DateTime(timezone=True), nullable=False, server_default=sa.text('now()')),
    sa.Column('resource_usage', postgresql.JSONB, nullable=True, server_default='{}'),
    sa.Column('metadata', postgresql.JSONB, nullable=True, server_default='{}'),
    sa.Column('created_at', sa.DateTime(timezone=True), nullable=False, server_default=sa.text('now()')),
    sa.Column('updated_at', sa.DateTime(timezone=True), nullable=False, server_default=sa.text('now()')),
)

# Create indexes for agent_states
op.create_index('idx_agent_state_agent', 'agent_states', ['agent_id'])
op.create_index('idx_agent_state_status', 'agent_states', ['current_status'])
op.create_index('idx_agent_state_activity', 'agent_states', ['last_activity'])

# Create agent_capabilities table
op.create_table(
    'agent_capabilities',
    sa.Column('id', postgresql.UUID(as_uuid=True), primary_key=True),
    sa.Column('agent_id', postgresql.UUID(as_uuid=True), sa.ForeignKey('agents.id', ondelete='CASCADE'), nullable=F
    sa.Column('capability_name', sa.String(255), nullable=False),
    sa.Column('capability_version', sa.String(50), nullable=False, server_default='1.0.0'),
    sa.Column('enabled', sa.Boolean, nullable=False, server_default='true'),
    sa.Column('config', postgresql.JSONB, nullable=True, server_default='{}'),
    sa.Column('description', sa.Text, nullable=True),
    sa.Column('created_at', sa.DateTime(timezone=True), nullable=False, server_default=sa.text('now()')),
    sa.Column('updated_at', sa.DateTime(timezone=True), nullable=False, server_default=sa.text('now()')),
)

# Create indexes for agent_capabilities
```

```
    op.create_index('idx_agent_cap_agent', 'agent_capabilities', ['agent_id'])
    op.create_index('idx_agent_cap_name', 'agent_capabilities', ['capability_name'])
    op.create_index('idx_agent_cap_enabled', 'agent_capabilities', ['enabled'])
    op.create_index('idx_agent_cap_agent_name', 'agent_capabilities', ['agent_id', 'capability_name'])
    op.create_unique_constraint('uq_agent_capability', 'agent_capabilities', ['agent_id', 'capability_name'])

    # Create agent_metrics table
    op.create_table(
        'agent_metrics',
        sa.Column('id', postgresql.UUID(as_uuid=True), primary_key=True),
        sa.Column('agent_id', postgresql.UUID(as_uuid=True), sa.ForeignKey('agents.id', ondelete='CASCADE'), nullable=F
        sa.Column('metric_name', sa.String(255), nullable=False),
        sa.Column('metric_value', sa.Float, nullable=False),
        sa.Column('metric_type', postgresql.ENUM(name='metrictype'), nullable=False, server_default='gauge'),
        sa.Column('tags', postgresql.JSONB, nullable=True, server_default='{}'),
        sa.Column('recorded_at', sa.DateTime(timezone=True), nullable=False, server_default=sa.text('now()')),
    )

    # Create indexes for agent_metrics
    op.create_index('idx_agent_metric_agent', 'agent_metrics', ['agent_id'])
    op.create_index('idx_agent_metric_name', 'agent_metrics', ['metric_name'])
    op.create_index('idx_agent_metric_recorded', 'agent_metrics', ['recorded_at'])
    op.create_index('idx_agent_metric_agent_name', 'agent_metrics', ['agent_id', 'metric_name'])
    op.create_index('idx_agent_metric_agent_recorded', 'agent_metrics', ['agent_id', 'recorded_at'])


def downgrade():
    """Drop agent state tables"""

    # Drop tables in reverse order
    op.drop_table('agent_metrics')
    op.drop_table('agent_capabilities')
    op.drop_table('agent_states')
    op.drop_table('agent_configs')

    # Drop ENUM types
    op.execute("""
        DROP TYPE IF EXISTS metrictype;
        DROP TYPE IF EXISTS agentstatusdetail;
        DROP TYPE IF EXISTS configtype;
    """)
```

✅ **FILE 4 COMPLETE (~150 lines)**

📝 QUICK VERIFICATION (Part 1)

After creating these 4 files, verify they work:

Step 1: Check Files Exist

bash

```bash
cd ~/optiinfra

# Check new model file
ls -la shared/database/models/agent_state.py

# Check modified files
ls -la shared/database/models/core.py
ls -la shared/database/models/__init__.py

# Check migration
ls -la shared/database/migrations/versions/002_agent_state_tables.py
```

Step 2: Test Python Imports

bash

```bash
cd ~/optiinfra

# Test if new models can be imported
python -c "from shared.database.models.agent_state import AgentConfig, AgentState, AgentCapability, AgentMetric; print

# Expected output:
# ✅ Models import successfully
```

Step 3: Run Alembic Migration

bash

```bash
cd ~/optiinfra/shared/database

# Check current migration state
alembic current
# Expected: 001_core_schema (current)

# Run the new migration
alembic upgrade head

# Expected output:
# INFO  [alembic.runtime.migration] Running upgrade 001_core_schema -> 002_agent_state_tables, Create agent state t
```

Step 4: Verify Tables Created

bash

```
# Connect to PostgreSQL
psql postgresql://optiinfra:password@localhost:5432/optiinfra

# List all tables (should now be 10 tables)
\dt

# Expected output (6 from 0.2a + 4 new):
#  agent_configs
#  agent_states
#  agent_capabilities
#  agent_metrics
#  customers
#  agents
#  events
#  recommendations
#  approvals
#  optimizations

# Describe agent_configs table
\d agent_configs

# Expected: Should show all columns, indexes, foreign key to agents

# Exit
\q
```

🎯 PART 1 SUMMARY

Files Created/Modified:

✅ **shared/database/models/agent_state.py** - 4 new models (~400 lines)
✅ **shared/database/models/core.py** - Modified (added relationships)
✅ **shared/database/models/init.py** - Modified (exports)
✅ **shared/database/migrations/versions/002_agent_state_tables.py** - Migration (~150 lines)

What Works Now:

✅ Can import all new models in Python
✅ Can run Alembic migration
✅ 4 new tables created in PostgreSQL
✅ All foreign keys to agents table working
✅ All indexes created

What's in Part 2:

- Seed data (configs, states, capabilities, metrics for test agents)
- Test fixtures
- Test cases (10+ comprehensive tests)
- All validation commands
- Troubleshooting guide
- Success criteria checklist
- Git commit instructions

⏭️ NEXT STEP

Download PART 2 to get:

- Complete seed data implementation
- Full test suite
- Step-by-step validation
- Troubleshooting
- Final checklist

File name: `FOUNDATION-0.2b-Agent-State-Tables-PART2-Testing.md`

📥 HOW TO DOWNLOAD THIS FILE

1. Click the **"Copy"** dropdown button at the top of this artifact
2. Select **"Download as txt"**
3. Save as: `FOUNDATION-0.2b-Agent-State-Tables-PART1-Code.md`

✅ PART 1 COMPLETE! Ready for PART 2?